社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－HPC－111 (16)
2007／8／1

タスク並列スクリプト言語のビジュアル開発環境の構築

谷口和也[†] 松本真樹[†,☆] 大野和彦[†]
佐々木敬泰[†] 近藤利夫[†] 中島浩[††]

既存の並列プログラミング言語処理系には設計からデバッグまでを一貫して行うことができる実用的な統合開発環境が存在しない．そのため，特に大規模な並列プログラミングは容易ではない．そこで，本研究では我々が開発している並列スクリプト言語 MegaScript を対象として実用的な統合開環境を構築することを目指している．本稿ではその一部であるプログラミング支援を目的としたビジュアル開発環境について述べる．本環境は，並列処理の流れを表すタスクネットワークを直接描くことのできるエディタと，コードを記述するエディタで構成され，可視化モデルによるビジュアル開発とコードによる開発を併用できる．

Visual Development Environment for a Task Parallel Script Language

Kazuya Taniguchi,[†] Masaki Matsumoto,[†,☆] Kazuhiko Ohno,[†]
Takahiro Sasaki,[†] Toshio Kondo[†] and Hiroshi Nakashima[††]

Although integrated development environment (IDE) is common in sequential programming, such environment does not exist for parallel programming. This makes large parallel programs difficult to write. So we are developing an IDE for our parallel script language called Megascript. In this paper we describe a visual environment which is part of our IDE. This visual development environment provides both task network editor for visual editing the task process flow and a code editor for editing code text.

1. はじめに

近年，複雑な物理計算を要する分野においては，Pflops 以上の計算能力が求められており，100 万台規模のプロセッサを用いたメガスケールコンピューティングが必要となった．そのため，メガスケール規模での並列処理向けのタスク並列スクリプト言語 MegaScript[1] が開発されている．

MegaScript は，逐次や並列の外部プログラムを実行単位であるタスクとして扱い，各タスクをつなぐ通信路をストリームとして扱う．そして，それら複数のタスクとストリームをつないで組み合わせた，タスクネットワークモデルとしてその形状を記述し，並列処理を行わせる．

しかし，現在 MegaScript には，Delphi[2] のような RAD(Rapid Application Development) 技法に基づくプログラムの開発環境が存在せず，大規模な並列プログラムを効率的に作成するのが難しい．また，MegaScript に限らず並列プログラミング環境において，高性能 GridRPC アプリケーションの開発環境[3] のようなデバッグ支援のための情報可視化ツールは存在するが，コーディングからデバッグまでの開発全体を支援する統合開発環境のようなものは存在していない．

そこで，本研究では並列言語の実用的な統合開発環境を構築し，並列プログラムを開発しやすくすることを目指す．MegaScript はタスクネットワークモデルを記述する言語であるため，タスクネットワークモデルを直接視覚的に描くことができれば開発効率が向上する．また，可視化モデルを直接編集するだけではなく，従来通りのテキストによる記述も可能にすることで，他の環境上で開発したコードを利用することが可能になり，コードで記述しやすい場合はテキスト入力することで，さらに効率があがることが期待できる．そこで，可視化した並列モデルを描くことでプログラムコードが生成され，また，そのコードを修正すると

† 三重大学
Mie University
☆ 現在，株式会社 科学情報システムズ
Presently with Science Information Systems
†† 京都大学
Kyoto University

並列モデルが修正される，可視化モデルとコードを併用可能な開発環境を提案する．

以下，次章で MegaScript の概要を述べ，３章では本研究が目指す MegaScript 統合開発環境について述べる．４章および５章では，今回実装した MegaScript ビジュアル開発環境の設計および実装について述べる．６章では今後の展望について述べ，最後にまとめとする．

2. タスク並列スクリプト言語 MegaScript

2.1 言語の概要

2.1.1 タ ス ク

タスクは MegaScript の並列実行単位である．タスクは任意の言語で記述された独立性の高い逐次または並列の外部プログラムであり，内部の処理に関しては MegaScript からはブラックボックスとなっている．

2.1.2 ストリーム

タスク間の通信は，タスクの標準出力の内容を他のタスクの標準入力につないだ通信路によって行われ，この通信路を MegaScript ではストリームと呼んでいる．ストリームには複数のタスクを接続することが可能であり，一対多，多対多などの通信を容易に実現可能である．入力端に複数のタスクを接続した場合，メッセージは非決定的にマージされる．出力端に複数のタスクを接続した場合は，メッセージはそれらのタスクにマルチキャストされる．

2.1.3 プログラミングモデル

MegaScript では，これらタスクやストリームを表すオブジェクトを定義・生成し，それらを組み合わせたタスクネットワークモデルの形状を記述する (図 1,2)．計算の主要部分は外部プログラムとして用意されるため，MegaScript では並列実行に関する制御情報を記述する．このため，実行効率は高くないが記述性が優位なスクリプト言語とし，ベース言語にはオブジェクト指向スクリプト言語の Ruby[4] を用いている．

2.2 プログラミングの流れ

MegaScript でプログラミング作業を行う際には，以下のような流れをとることになる．

(1) タスクネットワークの設計
(2) タスクネットワークを実現するコードを API メソッドを用いて記述
  (a) タスク・ストリームの定義
  (b) タスクストリームの接続
  (c) タスク・ストリームの生成

2.3 現状の課題

MegaScript は，大規模並列処理に対応するために

```
1  p = Task.new_array(N, "producer")
2  c = Task.new("consumer")
3  s = Stream.new
4  s.connect(p, IN)
5  s.connect(c, OUT)
6  p.create(1..N)
7  c.create
8  s.create
```

図 1 タスクネットワークの定義例

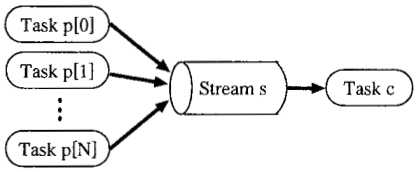

図 2 タスクネットワーク

API に TaskArray/StreamArray クラスを用意しており，１行で大量のオブジェクトの定義・接続・生成が可能である．

しかし，それでもなお大規模なものになると設計した並列モデルと等価なコードが記述できていないなどの人為的なミスが発生したり，コーディングの作業量が多くなってしまったりする可能性がある．このため，大規模な実用プログラムの開発を支援する開発環境が期待される．

3. MegaScript 統合開発環境

ソフトウェアの開発に関連するツールには，テキストエディタ・コンパイラ・デバッガなど様々なものが存在する．

統合開発環境は，これら従来ばらばらで利用していたプログラミングに必要なツールを一つのインタフェースで統合して利用できるようにしたものである．

現在，逐次プログラミング言語の処理系では，Delphi や Visual Basic[5] といった統合開発環境が存在しており，大規模な実用プログラムを効率よく開発することを可能にしている．しかし，並列プログラミング言語には KLIEG[6] や GridRPC アプリケーションの開発環境などの開発環境が存在するが，実用的な統合開発環境が存在しない．

そこで我々はプログラミング支援環境，実行環境，デバッグ環境などを統合した MegaScript の統合開発環境を構築し，並列プログラムを効率良く開発可能にすることを目指す．今回は，統合開発環境のプログラミング支援部分であるビジュアル開発環境を構築した．

3.1 ビジュアル開発環境

ビジュアル開発とは，従来プログラムをテキストに

よって記述していたものを，GUIを用いて視覚的な操作によって開発を行う手法のことである．そして，そのようなビジュアル開発の機能を備えたプログラミング支援環境をビジュアル開発環境と呼んでいる．

ビジュアル開発環境には大きく2つのタイプが存在し，コードを一切記述せずにプログラム全体をビジュアルに開発する環境と，ビジュアルに開発を行いやすい部分のみビジュアル開発を行い，残りは従来通りテキストによって記述する環境がある．

前者の代表的なものとしてビジュアル並列プログラミング環境のKLIEGや同じく並列ビジュアル環境であるVISTA[7]，逐次プログラミング言語処理系のPrograph[8]等がある．これらの環境では，設計図と類似した図式を直接プログラムとして編集する機能を提供しており，プログラムを直感的に作成することができる．そのため，プログラミング言語の文法を詳しく知らなくてもプログラムを作成することが可能であり，教育用などに用いられている．

一方欠点としては，全てビジュアルに開発するため，既存のコードや他の言語で記述されたコードの再利用を行うことが不可能であることが挙げられる．また，例えば，大量のタスクに対して少しずつ値を変えながら引数を設定する場合などは，ビジュアル開発よりコードでループ内にその規則を記述した方が容易に実現できる．このように，コード記述の方が容易である場合でもビジュアルに開発を行わなければならない．

このような点から，コードを一切記述せずプログラム全体をビジュアルに開発する環境は，実用的な開発環境として向いていない．

一方，一部のみビジュアルに開発するタイプとしてはDelphiやVisual Basicなどがある．これらの環境では，開発したいアプリケーションのビジュアルで開発しやすいGUI部分のみビジュアルに開発し，そのほかの部分は従来通りのコードで記述する仕組みをとっている．このような仕組みの場合，過去に他の環境上でコード記述により作成したプログラムを利用したり，GUI設計においてビジュアル開発しにくい場合にコードによる開発を行ったりすることが可能なため，開発が効率良く行え実用的であるといえる．しかしながら，一度ビジュアルで開発をした部分はコードで修正することはできず，またコードで記述した部分をビジュアルで修正することはできないといった制限が存在する．

3.2 MegaScript ビジュアル開発環境

MegaScriptは2章で述べたように，タスクネットワークを設計した後，それと等価なコードを記述する．従って，タスクネットワークをビジュアルに開発できる環境を用意することで，コードを記述する必要がなくなり，プログラマの負担を軽減することが可能になる．

また，MegaScriptのプログラムはタスク・ストリームの定義と接続関係を記述するという簡単なものである．そのため，ビジュアルに表現することが比較的容易なプログラミング言語であるといえる．DelphiやVisual Basicでは言語の特性上，ビジュアル開発とコード記述の双方向による開発が限られた範囲でしかサポートされていない．MegaScriptでは，計算の主要な部分はブラックボックスであり，タスクネットワークを記述することがメインであるという特性から，Delphiなどより相互反映部分を一般化でき，言語の大半の部分についてビジュアル開発とコード記述の両方をサポートすることが可能である．

そこで，MegaScript統合開発環境では，ビジュアル開発とコード記述を臨機応変に使い分けることが可能な開発環境を用意する．

ビジュアルで開発できる範囲が広いため，KLIEGやVISTAといった他のプログラム全体をビジュアルに開発するタイプの開発環境と似た面もあるが，コードによる記述をサポートしている点がそれらと大きく異なる．ビジュアルとコード記述による双方向の開発を許すことで，並列プログラミング環境に今まで存在しなかった実用的な開発環境となることが期待される．

4. 設　　計

本ビジュアル開発環境では，ビジュアル開発と従来のコードによる記述をサポートするために，それぞれタスクネットワークを編集するグラフィカルなエディタとテキストエディタを用意し，これら2つのエディタの内容をインタラクティブに反映させる機能が必要になる．

MegaScriptビジュアル開発環境は，実用的な統合開発環境に向けて今後並列デバッガを利用した実行可視化ウィンドウを組み込んだり，さらなる拡張を予定しているため，この点も考えて設計を行わなければならない．

4.1 システムの実現方式

2つのエディタの内容をインタラクティブに反映させるには，一方のエディタの情報を他方のエディタの情報に変換する変換器が必要である．しかし，直接これらの情報を変換するような仕組みをとったとすると，既存のエディタがN個存在する際に，新たなエディタを追加しようとするとN個の変換器を新たに作らな

ければならなくなってしまい，実装コストが非常に高くなってしまう．

そのため，全てのエディタの情報を一度ある中間形式で表現し，それを経由して変換することにする．この方法をとると，N 個のエディタが存在している時でも，中間形式と新規エディタ間の 1 つの変換器を作ればよい．

そこで，MegaScript ビジュアル開発環境の構成は以下のようにする．

- 外部インタフェース
  タスクネットワークやコードエディタなど開発に必要なユーザインタフェース部分．
- 内部モデル
  開発中のプログラムの情報のマスタとなる．
- 変換器
  外部インタフェースと内部モデルのデータを変換する．
- パーサ
  コードエディタで入力されたコードの意味を内部モデルが認識するための解析器となる．修正が行われるたびに全てのコードをパースするのは非常に効率が悪いため，修正した行のみをパースする仕組みをとることにする．

4.2 相互変換

タスクネットワークエディタとコードエディタのデータをインタラクティブに変換するには，一方で変更したオブジェクトに対応するオブジェクトを知っていなければならない．本開発環境では内部モデルを経由して互いのデータを反映させるため，内部モデルが互いに対応するオブジェクトを把握できるような構造になっていなければならない．そのために，内部モデルは変換対象に対してリンクを持たせる必要がある．

また，実際のプログラムでは，ユーザが入力するコメントやスペース，タブの数などが含まれる．これらの情報は直接プログラムの実行とは関係がないが，タスクネットワークエディタで修正した場合にコードエディタでユーザが記述していたものを変更してしまうとユーザを混乱させ，使い勝手を悪くしてしまう．そこで内部モデルは，テキスト情報を保持し，修正時にこれらの情報が保持されるようにしている．

4.3 API メソッドの呼び出し間の依存関係の制御

MegaScript のコードには API メソッドの呼び出し間に依存関係が存在する．具体的には，new メソッドによりオブジェクトの生成を行い，その後オブジェクトを connect メソッドによって接続し，最後に create メソッドによってプロセスなどの実体を生成するという順序である．タスクネットワークエディタのようなビジュアルツールでの開発には，ユーザの開発操作の順序に制限はなく，最終的にユーザが思い描いたものが表示されれば良い．ところが，タスクネットワークエディタで開発した場合に，ユーザの操作順序通りにコードを自動生成してしまうと，依存関係を満たさず実行できないプログラムとなってしまう．例えば図 1 のコードの状態から，タスクネットワークエディタでタスク task_a を新しく配置し，ストリーム s と接続を行った場合を考える．このときタスクネットワークエディタでの開発順序通りにコードを挿入すると，図 3 のようになる．このコードは，8 行目のストリームの create の後に 11 行目で接続を行っていることなど，new や create メソッドの後に connect が行われているため，正しく実行できない．このような現象を防ぐには，自動生成されたコードが正しい順になるように並び替えなければならない．

```
1   p = Task.new_array(N, "producer")
2   c = Task.new("consumer")
3   s = Stream.new
4   s.connect(p, IN)
5   s.connect(c, OUT)
6   p.create(1..N)
7   c.create
8   s.create
9   task_a = Task.new("example")
10  task_a.create
11  s.connect(task_a, IN)
```

依存

図 3 ビジュアル開発の順序通り生成した場合のコード

ここで，プログラムを 3 つの領域に分割し，プログラム中に出現する全ての new，connect，create をそれぞれ対応する領域に配置すると，任意のオブジェクトについて new，connect，create が正しい順になることが保証できる．そこで，このような領域情報を保持し，新しく生成されたコードは対応する領域の最後に挿入する．図 3 の場合をこの方法で処理すると図 4 のようになり，正しいプログラムになる．このような挿入・削除がしやすいようにコード情報はリスト形式で保持する．

4.4 制御文の認識

これまでの設計により，単純なコードならば互いのエディタの情報を反映できる仕組みとなった．しかし，実際のプログラムにはループなどの制御文が含まれる．MegaScript では大規模並列処理のために，多数のオブジェクトに同じ引数を割り当てる際などループ文が使われることが多い．例えば，for ループ内に記述されたコードはループ回数分繰り返さなければならない．しかし，パーサで解析されるのは修正された行のみである．したがって，その行が for ループ内にあるのか

```
1  p = Task.new_array(N, "producer")
2  c = Task.new("consumer")
3  s = Stream.new
4  task_a = Task.new("example")
5  s.connect(p, IN)
6  s.connect(c, OUT)
7  s.connect(task_a, IN)
8  p.create(1..N)
9  c.create
10 s.create
11 task_a.create
```

図 4　各領域の最後に挿入した場合のコード

```
1  a = Task.new()
2  for i in 0..10
3      b[i] = Task.new()
4  end
```

図 5　制御ブロックによる for ループ内のコードの正しい認識

どうかを判断することができない．そのため内部モデルでは，最初に制御文が入力された時に制御文の範囲を記憶する．これを制御ブロックと呼ぶ (図 5 の点線部分)．この制御ブロックにより，任意の行についてどの制御文の内側にあるのかを知ることができる．

これにより，例えば図 5 の 3 行目をコードエディタで削除する際，この行は制御ブロックの内側にあることから，制御ブロック先頭の 2 行目をパースし直す．この結果，for ループ内で 10 回繰り返されていたことがわかり，タスクネットワークエディタの対応するタスクを 10 個削除して，正しく削除を反映することができる．

5. 実　　装

5.1 Eclipse

Eclipse[9] とは Java の統合開発環境である．しかし Eclipse は Java だけではなくプラグインという形で，さまざまな機能をユーザ独自に組み込むことが可能であり，Perl プログラム開発用の EPICd など様々なプラグインが存在している．プラグインとして実装することで，統合開発環境としての基本的なユーザインタフェースなどは Eclipse で既に使用可能であることや，GEF という描画エディタ作成用のフレームワークが用意されている等，開発環境の開発工数を削減することが可能となる．

5.2 本開発環境の実装方法

MegaScript のタスクは MegaScript からはブラックボックスであるが，足りないタスクを本統合開発環境上で C 等の他の言語で開発したいという状況が想定される．このような状況に対応するには，Eclipse では既に存在する C 等の開発環境のプラグインを組み込むだけでよく，拡張が行いやすい．開発環境の開発効率が良く，また拡張も行いやすいというメリットから，本ビジュアル開発環境はプラグインという形で Eclipse 上で動作するように実装した．

5.3 パーサの実装方法

MegaScript のパーサを初めから作るのは大変である．また，MegaScript にはもともと自身のコードを解析するためのトランスレータというものが存在する．そのため，パーサは MegaScript のトランスレータ部分を流用し，出力を開発環境用に少し改造した．MegaScript のトランスレータ部分は Racc[10] という Ruby 用のパーサジェネレータで作成されており，生成されたものは Ruby で動く．そのため，開発環境側から子プロセスとして Ruby を起動し動作させており，実行効率は良くない．この点は改善すべき点である．

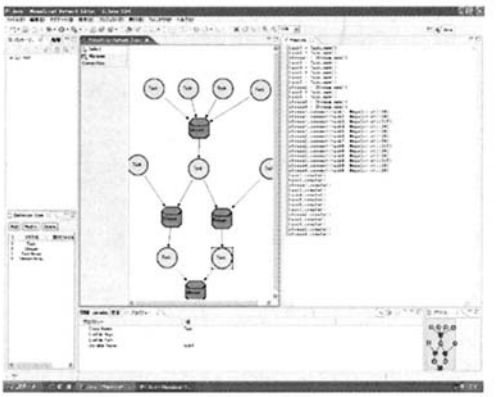

図 6　MegaScript 統合開発環境の実行画面

6. 今後の展望・課題

6.1 大規模並列モデルへの対応

MegaScript では大規模な並列処理を記述するため，タスクやストリームを 100 万規模で扱うことが想定される．しかし，このような規模のものを 1 つずつ扱っていては非常に効率が悪い．ビジュアル開発環境でも多数のオブジェクトを配置することによって視認性が悪くなるなど，同種の問題が起こることが想定される．

そのため，MegaScript ではタスクやストリームを大量に使用する際に，API の TaskArray/StreamArray クラスを用いることでユーザの負担を軽減している．そこで，ビジュアル開発環境でも TaskArray や StreamArray に対応することで表示や操作を簡単にする．

また，タスクネットワークエディタにおいて複数オブジェクトのグルーピング機能と階層化表示機能を取り入れることを検討している．前者は，複数のオブジェクトを1つにまとめるグルーピング機能を取り入れることで一度に表示するオブジェクトが減り，視認性の低下を防ぐことができる．またKLIEGのように，階層化の機能を取り入れることで，グルーピング機能と連携し，ネットワークトポロジの再利用が可能となり，開発効率の向上につながる．

6.2 並列デバッグ機能

現在，本研究室では並列デバッガの研究も行っている．GDB(The GNU Source-level Debugger)を少し改良し，各タスク毎にGDBを実行できるようにし，それを一括管理する仕組みの並列デバッガを作成中である．これにより，並列動作するタスク群に対して，コマンドを個別に，またはまとめて送信することを可能にしたり，情報を収集することが可能となる．これをタスクネットワークモデルでの可視化インタフェースにより，モデル上で指定したタスクにコマンドを送ったり，収集した各タスクの情報をモデル上に表示することで，どのタスクの情報か分かりやすく表示することが可能になる．この並列デバッガをMegaScript統合開発環境に統合することで，並列プログラムのデバッグ作業の効率が上がり，開発がさらに効率よく行えることが期待できる．

7. おわりに

本研究では，並列プログラミング言語における実用的な統合開発環境を構築し，大規模な並列プログラムを効率良く開発できるようにすることを目指している．本稿では，視覚化された並列モデルを作成することによる開発と従来通りのコードによる開発の両方をサポートし，それら互いのエディタがインタラクティブに反映されるタスク並列スクリプト言語のビジュアル開発環境を提案し，その設計・実装について述べた．

今回構築したビジュアル開発環境では，ユーザが臨機応変にタスクネットワークエディタとコードエディタを使い分けることで，開発を効率よく行うことが可能になる．

今後，並列デバッガなどを組込み，実用的な並列言語の統合開発環境を目指し機能を拡張していく．

謝辞　本研究の一部は文部科学省科学研究費補助金（特定領域研究，研究課題番号19024041,「高性能計算の高精度モデル化技術」）による．

参考文献

1) 大塚保紀, 深野佑公, 西里一史, 大野和彦, 中島浩: タスク並列スクリプト言語MegaScriptの構想, 先進的計算基盤システムシンポジウムSACSIS2003, pp. 73–76 (2003).
2) Ivan Hladni: Inside Delphi 2006 (Wordware Delphi Developer's Library), Wordware Publishing Inc.,(2005).
3) 小林孝嗣, 渡邊啓正, 本多弘樹: 高性能GridRPCアプリケーションの開発環境, 情報処理学会研究報告, HPC-104, pp. 1–6 (2005).
4) まつもとゆきひろ, 石塚圭樹: オブジェクト指向スクリプト言語Ruby, ASCII (1999).
5) Cary B. Shelly, Thomas J.Cashman, John F. Repede, Michael Mick: Microsoft Visual Basic 6: Introductory Concepts and Techniques (Shelly Cashman Series) , Course Technology Ptr(Sd), (1998).
6) 志築文太郎, 豊田正史, 高橋伸, 柴山悦哉: ビジュアル並列プログラミング環境KLIEG: プロセスネットワークパターンを利用した再利用性の向上と実行表示の効率化, ソフトウェア科学会WISS'96論文集, 近代科学社レクチャーノート, Vol. 16. pp. 81–90 (1996).
7) Stefan Schiffer and Joachim Hans Fröhlich: Visual Programming and Software Engineering with Vista. In Visual object-oriented programming: concepts and environments, chapter 10, pp. 199-227. Manning Publications Co., (1995).
8) P. T. Cox and T. Pietrzykowski : Using a Pictorial Representation to Combine Dataflow and Object-orientation in a Language Independent Programming Mechanism: IEEE,(1988).
9) Eclipse http://www.eclipse.org/
10) 青木峰郎: Rubyを256倍使うための本 無道編, 株式会社アスキー,(2001).